org-mode

IOANNIS ZANNOS

21. Dezember 2014

Inhaltsverzeichnis

1 binding for org show subtree

```
(eval-after-load 'org
   '(define-key org-mode-map (kbd "C-c C-x s") 'org-show-subtree))
```

2 Using ido for org-goto

```
(setq org-goto-interface 'outline-path-completion
      org-goto-max-level 10)
```

3 Working with icicles/ido-menu/lacarte in org-mode and elsewhere

3.1 lacarte/icicle-menu shortcut: H-C-i,

```
;; Previously bound only to org-mode map.
(global-set-key (kbd "H-TAB") 'icicle-imenu)
(global-set-key (kbd "H-C-l") 'lacarte-execute-menu-command)
```

3.2 making icicle-imenu and icicle-occur work with org-mode

Following needs review! Fri, Nov 28 2014, 10:44 EET

```
(defun org-icicle-occur ()
 "In org-mode, show entire buffer contents before running icicle-occur.
 Otherwise icicle-occur will not place cursor at found location,
 if the location is hidden."
  (interactive)
  (show-all)
  (icicle-occur (point-min) (point-max))
  (recenter 3))

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c '") 'org-edit-special))
(eval-after-load 'org
  '(define-key org-mode-map (kbd "H-i") 'org-icicle-occur))
(defun org-icicle-imenu (separate-buffer)
 "In org-mode, show entire buffer contents before running icicle-imenu.
Otherwise icicle-occur will not place cursor at found location,
if the location is hidden.
If called with prefix argument (C-u), then:
- open the found section in an indirect buffer.
- go back to the position where the point was before the command, in the
  original buffer."
  (interactive "P")
  (icicle-mode 1)
  (show-all)
  (let ((mark (point)))
    (icicle-imenu (point-min) (point-max) t)
    (cond (separate-buffer
           (org-tree-to-indirect-buffer)
           (goto-char mark))
          (t (recenter 4))))
  (icicle-mode -1))

(eval-after-load 'org
```

```
  '(define-key org-mode-map (kbd "C-c C-=") 'org-icicle-imenu))
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c i m") 'org-icicle-imenu))

;; install alternative for org-mode C-c = org-table-eval-formula
;; which is stubbornly overwritten by icy-mode.
(eval-after-load 'org
 '(define-key org-mode-map (kbd "C-c C-x =") 'org-table-eval-formula))

;; Both eval-after-load and org-mode hook do not work for switching off
;; prelude mode, whitespace.  So using shortcuts as workaround:

(defun turn-off-whitespace-mode ()
  (interactive)
  (whitespace-mode -1))

(defun turn-off-icicle-mode ()
  (interactive)
  (icicle-mode -1))

(defun turn-off-prelude-mode ()
  (interactive)
  (prelude-mode -1))

(global-set-key (kbd "H-x w") 'turn-off-whitespace-mode)
(global-set-key (kbd "H-x p") 'turn-off-prelude-mode)
(global-set-key (kbd "H-x i") 'turn-off-icicle-mode)

(add-hook 'org-mode-hook
          (lambda ()
           (local-set-key (kbd "C-c M-=") 'org-table-eval-formula)
            (local-set-key (kbd "C-c '") 'org-edit-special)))

;;; ???? Adapt org-mode to icicle menus when refiling (C-c C-w)
;;; Still problems. Cannot use standard org refiling with icicles activated!
(setq org-outline-path-complete-in-steps nil)
```

3.3 Definitely switch prelude off in org mode, as it totally screws-up key bindings

Especially in the case of Meta-shift-up and Meta-shift-down for spreadsheets. Have not figured out yet how to override those keys specifically.

```
(add-hook 'org-mode-hook
          (lambda ()
            (prelude-mode -1)))
```

```
(add-hook 'org-mode-hook 'prelude-off)
```

3.4 Providing alternatives for refile and copy using icicles

```
(defun org-refile-icy (as-subtree &optional do-copy-p)
  "Alternative to org-refile using icicles.
Refile or copy current section, to a location in the file selected with icicles.
Without prefix argument: Place the copied/cut section it *after* the selected section.
With prefix argument: Make the copied/cut section *a subtree* of the selected section.

Note 1: If quit with C-g, this function will have removed the section that
is to be refiled.  To get it back, one has to undo, or paste.

Note 2: Reason for this function is that icicles seems to break org-modes headline
buffer display, so onehas to use icicles for all headline navigation if it is loaded."
  (interactive "P")
  (outline-back-to-heading)
  (if do-copy-p (org-copy-subtree) (org-cut-subtree))
  (show-all)
  (icicle-imenu (point-min) (point-max) t)
  (outline-next-heading)
  (unless (eq (current-column) 0) (insert "\n"))
  (org-paste-subtree)
  (if as-subtree (org-demote-subtree)))

(defun org-copy-icy (as-subtree)
 "Copy section to another location in file, selecting the location with icicles.
See org-refile-icy."
  (interactive "P")
  (org-refile-icy as-subtree t))

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c i r") 'org-refile-icy))
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c i c") 'org-copy-icy))
```

4 Use visual line, whitespace and windmove in org-mode

```
(add-hook 'org-mode-hook 'visual-line-mode)
(add-hook 'org-mode-hook 'turn-off-whitespace-mode)
(add-hook 'org-shiftup-final-hook 'windmove-up)
(add-hook 'org-shiftleft-final-hook 'windmove-left)
(add-hook 'org-shiftdown-final-hook 'windmove-down)
(add-hook 'org-shiftright-final-hook 'windmove-right)
```

5 Customize Org-mode display, including todo colors

Adapted from:

```
(setq org-startup-indented t) ;; auto-indent text in subtrees
(setq org-hide-leading-stars t) ;; hide leading stars in subtree headings
(setq org-src-fontify-natively t) ;; colorize source-code blocks natively
(setq org-todo-keywords
      '((sequence
         "!!!(1)"  ; next action
         "!!(2)"  ; next action
         "!(3)"  ; next action
         "TODO(t)"  ; next action
         "STARTED(s)"
         "WAITING(w@/!)"
         "TOBLOG(b)"  ; next action
         "SOMEDAY(.)" "|"
         "DONE(x@/@)"
         "CANCELLED(c@)"
         "OBSOLETE(o@)")
        (sequence
         "TODELEGATE(-)"
         "DELEGATED(d)"
         "DELEGATE_DONE(1!)")))

(setq org-todo-keyword-faces
      '(("!!!" . (:foreground "red" :weight bold))
        ("!!" . (:foreground "tomato" :weight bold))
        ("!" . (:foreground "coral" :weight bold))
        ("TODO" . (:foreground "LightSalmon" :weight bold))
        ("TOBLOG" . (:foreground "MediumVioletRed" :weight bold))
        ("STARTED" . (:foreground "DeepPink" :weight bold))
        ("WAITING" . (:foreground "gold" :weight bold))
        ("DONE" . (:foreground "SeaGreen" :weight bold))
        ("CANCELLED" . (:foreground "wheat" :weight bold))
        ("OBSOLETE" . (:foreground "CadetBlue" :weight bold))
        ("TODELEGATE" . (:foreground "DeepSkyBlue" :weight bold))
        ("DELEGATED" . (:foreground "turquoise" :weight bold))
        ("DELEGATE_DONE" . (:foreground "LawnGreen" :weight bold))
        ("WAITING" . (:foreground "goldenrod" :weight bold))
        ("SOMEDAY" . (:foreground "gray" :weight bold))))
```

6 Mobile Org

```
;; the rest of the setup was done by customizing the variables
```

```
;; org-mobile-directory and org-mobile-inbox-for-pull, and is in custom.el

(global-set-key (kbd "H-h m p") 'org-mobile-push)
(global-set-key (kbd "H-h m l") 'org-mobile-pull)
```

Following was tested, works OK, but is disabled for the moment:
`http://kenmankoff.com/2012/08/17/emacs-org-mode-and-mobileorg-auto-sync/`

```
(defun install-monitor (file secs)
  (run-with-timer
   0 secs
   (lambda (f p)
    (unless (< p (second (time-since (elt (file-attributes f) 5))))
       (org-mobile-pull)))
   file secs))

(defvar monitor-timer
 (install-monitor (concat org-mobile-directory "/mobileorg.org") 30)
  "Check if file changed every 30 s.")
```

7 line->headline

```
(defun org-headline-line ()
  "convert current line into headline at same level as above."
  (interactive)
  (beginning-of-line)
  (org-meta-return)
  (delete-char 1))

(eval-after-load 'org
  '(progn
    (define-key org-mode-map (kbd "C-M-<return>") 'org-headline-line)))
```

8 Agenda

8.1 Global key for org-agenda: C-c a

```
(global-set-key "\C-ca" 'org-agenda)
```

8.2 Add, remove, save agenda file list

```
(defvar org-agenda-list-save-path
  "~/.emacs.d/savefile/org-agenda-list.el"
"Path to save the list of files belonging to the agenda.")
```

```
(defun org-agenda-save-file-list ()
  "Save list of desktops from file in org-agenda-list-save-path"
  (interactive)
  (save-excursion
    (let ((buf (find-file-noselect org-agenda-list-save-path)))
      (set-buffer buf)
      (erase-buffer)
      (print (list 'quote org-agenda-files) buf)
      (save-buffer)
      (kill-buffer)
    (message "org-agenda file list saved to: %s" org-agenda-list-save-path))))

(defun org-agenda-load-file-list ()
  "Load list of desktops from file in org-agenda-list-save-path"
  (interactive)
  (save-excursion
    (let ((buf (find-file-noselect org-agenda-list-save-path)))
      (set-buffer buf)
      (setq org-agenda-files (eval (read (buffer-string))))
      (kill-buffer)
    (message "org-agenda file list loaded from: %s" org-agenda-list-save-path))))

(defun org-agenda-add-this-file-to-agenda ()
  "Add the file from the current buffer to org-agenda-files list."
  (interactive)
  (let (path)
   ;; (org-agenda-file-to-front) ;; adds path relative to user home dir
    ;; (message "Added current buffer to agenda files.")
    (let ((path (buffer-file-name (current-buffer))))
      (cond (path
        (add-to-list 'org-agenda-files path)
        (org-agenda-save-file-list)
        (message "Added file '%s' to agenda file list"
                 (file-name-base path)))
        (t (message "Cannot add buffer to file list. Save buffer first."))))))

(defun org-agenda-remove-this-file-from-agenda (&optional select-from-list)
  "Remove a file from org-agenda-files list.
If called without prefix argument, remove the file of the current buffer.
If called with prefix argument, then select a file from org-agenda-files list."
  (interactive "P")
  (let (path)
   (if select-from-list
       (let  ((menu (grizzl-make-index org-agenda-files)))
      (setq path (grizzl-completing-read "Choose an agenda file: " menu)))
     (setq path (buffer-file-name (current-buffer))))
```

```
   (setq org-agenda-files
      (remove (buffer-file-name (current-buffer)) org-agenda-files)))
  (org-agenda-save-file-list)
  (message "Removed file '%s' from agenda file list"
           (file-name-base (buffer-file-name (current-buffer)))))

(defun org-agenda-open-file ()
  "Open a file from the current agenda file list."
  (interactive)
  (let* ((menu (grizzl-make-index org-agenda-files))
     (answer (grizzl-completing-read "Choose an agenda file: " menu)))
    (find-file answer)))

(defun org-agenda-list-files ()
  "List the paths that are currently in org-agenda-files"
  (interactive)
  (let  ((menu (grizzl-make-index org-agenda-files)))
   (grizzl-completing-read "These are currently the files in list org-agenda-files. " menu)))

(defun org-agenda-list-menu ()
 "Present menu with commands for loading, saving, adding and removing
files to org-agenda-files."
 (interactive)
 (let* ((menu (grizzl-make-index
               '("org-agenda-save-file-list"
                 "org-agenda-load-file-list"
                 "org-agenda-list-files"
                 "org-agenda-open-file"
                 "org-agenda-add-this-file-to-agenda"
                 "org-agenda-remove-this-file-from-agenda")))
      (command (grizzl-completing-read "Choose a command: " menu)))
   (call-interactively (intern command))))

(global-set-key (kbd "H-a H-a") 'org-agenda-list-menu)
```

8.3 Calendar framework: Show org agenda in iCal-style layout

```
(require 'calfw-org)
```

8.4 Global key for cfw org calendar framework): C-c M-a

```
(global-set-key "\C-c\M-a" 'cfw:open-org-calendar)
(global-set-key "\C-c\C-xm" 'org-mark-ring-goto)
```

8.5 Insert DATE property

```
(defun org-set-date (&optional active property)
  "Set DATE property with current time.  Active timestamp."
  (interactive "P")
  (org-set-property
   (if property property "DATE")
   (cond ((equal active nil)
       (format-time-string (cdr org-time-stamp-formats) (current-time)))
         ((equal active '(4))
          (concat "["
                  (substring
             (format-time-string (cdr org-time-stamp-formats) (current-time))
                   1 -1)
                  "]"))
         ((equal active '(16))
          (concat
           "["
           (substring
        (format-time-string (cdr org-time-stamp-formats) (org-read-date t t))
            1 -1)
           "]"))
         ((equal active '(64))
       (format-time-string (cdr org-time-stamp-formats) (org-read-date t t))))))

;; Note: This keybinding is in analogy to the standard keybinding:
;; C-c . -> org-time-stamp
(eval-after-load 'org
  '(progn
     (define-key org-mode-map (kbd "C-c C-.") 'org-set-date)
     ;; Prelude defines C-c d as duplicate line
    ;; But we disable prelude in org-mode because of other, more serious conflicts,
     ;; So we keep this alternative key binding:
     (define-key org-mode-map (kbd "C-c d") 'org-set-date)))
```

8.6 Set DUE property with selected time/date

```
(defun org-set-due-property ()
  (interactive)
  (org-set-property
   "DUE"
  (format-time-string (cdr org-time-stamp-formats) (org-read-date t t))))

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c M-.") 'org-set-due-property))
```

9 Class and Project notes, tags, diary

```
(setq org-tag-alist
      '(
        ("home" . ?h)
        ("finance" . ?f)
        ("eastn" . ?e)
        ("avarts" . ?a)
        ("erasmus" . ?E)
        ("researchfunding" . ?r)))

(defvar iz-log-dir
  (expand-file-name
   "~/Dropbox/000WORKFILES/")
 "This directory contains all notes on current projects and classes")

(setq diary-file (concat iz-log-dir "PRIVATE/diary"))

(defadvice org-agenda (before update-agenda-file-list ())
 "Re-createlist of agenda files from contents of relevant directories."
  (iz-update-agenda-file-list)
  (icicle-mode 1))

(defadvice org-agenda (after turn-icicles-off ())
 "Turn off icicle mode since it interferes with some other keyboard shortcuts."
  (icicle-mode -1))

(ad-activate 'org-agenda)

(defadvice org-refile (before turn-icicles-on-for-refile ())
 "Re-createlist of agenda files from contents of relevant directories."
  (icicle-mode 1))

(defadvice org-refile (after turn-icicles-off-for-refile ())
 "Turn off icicle mode since it interferes with some other keyboard shortcuts."
  (icicle-mode -1))

(ad-activate 'org-refile)

(defun iz-update-agenda-file-list ()
 "Set value of org-agenda-files from contents of relevant directories."
  (setq org-agenda-files
      (let ((folders (file-expand-wildcards (concat iz-log-dir "*")))
          (files (file-expand-wildcards (concat iz-log-dir "*.org"))))
           (dolist (folder folders)
             (setq files
```

```
                  (append
                   files
                (file-expand-wildcards (concat folder "/*.org")))))
          (-reject
           (lambda (f)
             (string-match-p "/\\." f))
           files)))
  (message "the value of org-agenda-files was updated"))

(defvar iz-last-selected-file
  nil
  "Path of file last selected with iz-org-file menu.
Used to refile to date-tree of last selected file.")

(defun iz-goto-last-selected-file ()
  (interactive)
  (if iz-last-selected-file
      (find-file iz-last-selected-file)
    (iz-find-file)))

(defun iz-refile-to-date-tree (&optional use-last-selected)
  "Refile to last selected file, using DATE timestamp
to move to file-datetree."
  (interactive "P")
  (let ((origin-buffer (current-buffer))
        (origin-filename (buffer-file-name (current-buffer)))
        (date (calendar-gregorian-from-absolute
               (org-time-string-to-absolute
                (or (org-entry-get (point) "CLOSED")
                 (org-entry-get (point) "DATE"))))))
    (org-cut-subtree)
    (if (and iz-last-selected-file use-last-selected)
        (find-file iz-last-selected-file)
      (iz-find-file))
    (org-datetree-find-date-create date)
    (move-end-of-line nil)
    (open-line 1)
    (next-line)
    (org-paste-subtree 4)
    (save-buffer)
    (find-file origin-filename)))

(defun org-process-entry-from-mobile-org ()
  (interactive)
  (org-back-to-heading 1)
  (next-line 1)
```

```
  (let ((time (cadr (org-element-timestamp-parser))))
    (org-entry-put nil "DATE" (plist-get time :raw-value)))
  (outline-next-heading))

(defun iz-get-and-refile-mobile-entries ()
  (interactive)
 (org-mobile-pull)
 (let* ((mobile-file (file-truename "~/org/from-mobile.org"))
        (mobile-buffer (find-file mobile-file))
     (log-buffer (find-file (concat iz-log-dir "PRIVATE/LOG.org"))))
   (with-current-buffer
       mobile-buffer
     (org-map-entries
      (lambda ()
        (let* ((timestamp
              (cdr (assoc "TIMESTAMP_IA" (org-entry-properties))))
               (date
               (calendar-gregorian-from-absolute
                (org-time-string-to-absolute timestamp))))
          (org-copy-subtree)
          (with-current-buffer
              log-buffer
            (org-datetree-find-date-create date)
            (move-end-of-line nil)
            (open-line 1)
            (next-line)
            (org-paste-subtree 4)
            (org-set-property "DATE" (concat "<" timestamp ">"))
            (org-set-tags-to ":mobileorg:"))))))
   (copy-file
    mobile-file
    (concat
     (file-name-sans-extension mobile-file)
     (format-time-string "%Y-%m-%d-%H-%M-%S")
     ".org"))
   (with-current-buffer
       mobile-buffer
     (erase-buffer)
     (save-buffer))))

(defun iz-refile-notes-to-log ()
  (interactive)
 (let* ((notes-file (concat iz-log-dir "NOTES/notes.org"))
        (notes-buffer (find-file notes-file))
     (log-buffer (find-file (concat iz-log-dir "PRIVATE/LOG.org"))))
   (with-current-buffer
```

```
      notes-buffer
     (org-map-entries
      (lambda ()
        (let* ((timestamp (org-entry-get (point) "DATE"))
               (date
               (calendar-gregorian-from-absolute
                (org-time-string-to-absolute timestamp))))
          (org-copy-subtree)
          (with-current-buffer
              log-buffer
            (org-datetree-find-date-create date)
            (move-end-of-line nil)
            (open-line 1)
            (next-line)
            (org-paste-subtree 4)
          (org-set-property "DATE" (concat "<" timestamp ">")))))))
   (copy-file
    notes-file
    (concat
     (file-name-sans-extension notes-file)
     (format-time-string "%Y-%m-%d-%H-%M-%S")
     ".org"))
   (with-current-buffer
       notes-buffer
     (erase-buffer)
     (save-buffer))))

(defun iz-insert-file-as-snippet ()
  (interactive)
  (insert-file-contents (iz-select-file-from-folders)))

(defun iz-select-file-from-folders ()
  (iz-org-file-menu (iz-select-folder)))

(defun iz-select-folder ()
  (let*
      ((folders (-select 'file-directory-p
                         (file-expand-wildcards
                          (concat iz-log-dir "*"))))
       (folder-menu (grizzl-make-index
                     (mapcar 'file-name-nondirectory folders)))
    (folder (grizzl-completing-read "Select folder:" folder-menu)))
    folder))

(defun iz-org-file-menu (subdir)
  (let*
```

```
      ((files
     (file-expand-wildcards (concat iz-log-dir subdir "/[a-zA-Z0-9]*.org")))
       (projects (mapcar 'file-name-sans-extension
                         (mapcar 'file-name-nondirectory files)))
       (dirs
        (mapcar (lambda (dir)
                  (cons (file-name-sans-extension
                                (file-name-nondirectory dir)) dir))
                files))
       (project-menu (grizzl-make-index projects))
     (selection (cdr (assoc (grizzl-completing-read "Select file: " project-menu)
                            dirs))))
    (setq iz-last-selected-file selection)
    selection))

(defun iz-get-refile-targets ()
  (interactive)
 (setq org-refile-targets '((iz-select-file-from-folders . (:maxlevel . 2)))))

(defun iz-find-file (&optional dired)
  "open a file by selecting from subfolders."
  (interactive "P")
  (cond ((equal dired '(4))
         (dired (concat iz-log-dir (iz-select-folder))))
        ((equal dired '(16)) (dired iz-log-dir))
        ((equal dired '(64))
         (dirtree (concat iz-log-dir (iz-select-folder)) nil))
        ((equal dired '(256))
         (dirtree iz-log-dir nil))
        (t
         (find-file (iz-select-file-from-folders))
         (goto-char 0)
         (if (search-forward "*# -*- mode:org" 100 t)
             (org-decrypt-entries)))))

;; Following needed to avoid error message ls does not use dired.
(setq ls-lisp-use-insert-directory-program nil)
(require 'ls-lisp)

(defun iz-open-project-folder (&optional open-in-finder)
  "Open a folder associated with a project .org file.
Select the file using iz-select-file-from-folders, and then open folder instead.
If the folder does not exist, create it."
  (interactive "P")
 (let ((path (file-name-sans-extension (iz-select-file-from-folders))))
    (unless  (file-exists-p path) (make-directory path))
```

```
   (if open-in-finder (open-folder-in-finder path) (dired path))))

(defvar iz-capture-keycodes "abcdefghijklmnoprstuvwxyzABDEFGHIJKLMNOPQRSTUVWXYZ1234567890.,(

(defun iz-log (&optional goto)
  "Capture log entry in date-tree of selected file."
  (interactive "P")
  (iz-make-log-capture-templates (iz-select-folder))
  (org-capture goto))

(defun iz-select-folder ()
  (let*
      ((folders (-select 'file-directory-p
                         (file-expand-wildcards
                          (concat iz-log-dir "*"))))
       (folder-menu (grizzl-make-index
                     (mapcar 'file-name-nondirectory folders)))
     (folder (grizzl-completing-read "Select folder:" folder-menu)))
    (file-name-nondirectory folder)))

(defun iz-make-log-capture-templates (subdir)
 "Make capture templates for selected subdirectory under datetree."
 (setq org-capture-templates
       (setq org-capture-templates
             (let* (
                    (files
                     (file-expand-wildcards
                  (concat iz-log-dir subdir "/[a-zA-Z0-9]*.org")))
                 (projects (mapcar 'file-name-nondirectory files))
                    (dirs
              (mapcar (lambda (dir) (cons (file-name-sans-extension
                                           (file-name-nondirectory dir))
                                          dir))
                              files)))
               (-map-indexed (lambda (index item)
                               (list
                     (substring iz-capture-keycodes index (+ 1 index))
                                 (car item)
                                 'entry
                                 (list 'file+datetree (cdr item))
                     "* %?\n :PROPERTIES:\n :DATE:\t%T\n :END:\n\n%i\n"))
                             dirs)))))

(defun iz-todo (&optional goto)
  "Capture TODO entry in date-tree of selected file."
  (interactive "P")
```

```
  (iz-make-todo-capture-templates (iz-select-folder))
  (org-capture goto))

(defun iz-make-todo-capture-templates (subdir)
  "Make capture templates for project files"
 (setq org-capture-templates
       (setq org-capture-templates
             (let* (
                    (files
                     (file-expand-wildcards
                   (concat iz-log-dir subdir "/[a-zA-Z0-9]*.org")))
                   (projects (mapcar 'file-name-nondirectory files))
                      (dirs
              (mapcar (lambda (dir) (cons (file-name-sans-extension
                                           (file-name-nondirectory dir))
                                          dir))
                      files)))
               (-map-indexed
                (lambda (index item)
                  (list
                   (substring iz-capture-keycodes index (+ 1 index))
                    (car item)
                    'entry
                    (list 'file+headline (cdr item) "TODOs")
                "* TODO %?\n :PROPERTIES:\n :DATE:\t%U\n :END:\n\n%i\n"))
                dirs)))))

(defun iz-goto (&optional level)
  (interactive "P")
  (if level
    (setq org-refile-targets (list (cons (iz-select-file-from-folders) (cons :level level))))
   (setq org-refile-targets (list (cons (iz-select-file-from-folders) '(:maxlevel . 3)))))
  (org-refile '(4)))

(defun iz-refile (&optional goto)
  "Refile to selected file."
  (interactive "P")
 (setq org-refile-targets (list (cons (iz-select-file-from-folders) '(:maxlevel . 3))))
  (org-refile goto))

(defun iz-org-file-command-menu ()
  "Menu of commands operating on iz org files."
(interactive)
  (let* ((menu (grizzl-make-index
                '(
                  "iz-log"
```

```
                    "iz-todo"
                    "iz-refile-to-date-tree"
                    "iz-refile"
                    "iz-open-project-folder"
                    "iz-find-file"
                    "iz-goto"
                    "iz-goto-last-selected-file"
                    "org-agenda"
                    "iz-get-and-refile-mobile-entries"
                    "iz-refile-notes-to-log"
                    "iz-insert-file-as-snippet"
                    )))
      (selection (grizzl-completing-read "Select command: " menu)))
    (eval (list (intern selection)))))

(global-set-key (kbd "H-h H-m") 'iz-org-file-command-menu)
(global-set-key (kbd "H-h H-h") 'iz-org-file-command-menu)
(global-set-key (kbd "H-h H-f") 'iz-find-file)
(global-set-key (kbd "H-h H-d") 'iz-open-project-folder)
(global-set-key (kbd "H-h H-l") 'iz-log)
(global-set-key (kbd "H-h L") 'iz-goto-last-selected-file)
(global-set-key (kbd "H-h H-i") 'iz-insert-file-as-snippet)
(global-set-key (kbd "H-h H-t") 'iz-todo)
(global-set-key (kbd "H-h H-r") 'iz-refile)
(global-set-key (kbd "H-h r") 'iz-refile-to-date-tree)
(global-set-key (kbd "H-h H-g") 'iz-goto)
(global-set-key (kbd "H-h H-c H-w") 'iz-refile)
(global-set-key (kbd "H-h H-c H-a") 'org-agenda)

;; Experimental:
(defun iz-make-finance-capture-template ()
  (setq org-capture-templates
        (list
         (list
          "f" "FINANCE"
          'entry
       (list 'file+datetree (concat iz-log-dir "projects/FINANCE.org"))
       "* %^{title}\n :PROPERTIES:\n :DATE:\t%T\n :END:\n%^{TransactionType}p%^{category}p%^{
          ))))
```

10 Org-Babel

10.1 Org-Babel: enable some languages

Enable some cool languages in org-babel mode.

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((emacs-lisp . t)
   (sh . t)
   (ruby . t)
   (python . t)
   (perl . t)
   ))
```

10.2 Org-Babel: load current file

```
(defun org-babel-load-current-file ()
  (interactive)
  (org-babel-load-file (buffer-file-name (current-buffer))))

;; Note: Overriding default key binding to provide consistent pattern:
;; C-c C-v f -> tangle, C-c C-v C-f -> load
(eval-after-load 'org
 '(define-key org-mode-map (kbd "C-c C-v C-f") 'org-babel-load-current-file))
```

11 Orgmode latex customization

Note Mon, Dec 15 2014, 16:29 EET: XeLaTeX covers most needs that I have for western european languages, Greek and Japanese.

```
;;; Load latex package
(require 'ox-latex)

;;; Use xelatex instead of pdflatex, for support of multilingual fonts (Greek etc.)
;; Note: Use package polyglossia to customize dates and other details.
(setq org-latex-pdf-process
    (list "xelatex -interaction nonstopmode -output-directory %o %f"
         "xelatex -interaction nonstopmode -output-directory %o %f"
        "xelatex -interaction nonstopmode -output-directory %o %f"))

;; This is kept as reference. XeLaTeX covers all european/greek/asian needs.
;; It is the original setting for working with pdflatex:
;; (setq org-latex-pdf-process
;;  ("pdflatex -interaction nonstopmode -output-directory %o %f"
;;   "pdflatex -interaction nonstopmode -output-directory %o %f"
;;   "pdflatex -interaction nonstopmode -output-directory %o %f"))

;;; Add beamer to available latex classes, for slide-presentaton format
(add-to-list 'org-latex-classes
             '("beamer"
```

```
               "\\documentclass\[presentation\]\{beamer\}"
               ("\\section\{%s\}" . "\\section*\{%s\}")
               ("\\subsection\{%s\}" . "\\subsection*\{%s\}")
             ("\\subsubsection\{%s\}" . "\\subsubsection*\{%s\}")))

;;; Add memoir class (experimental)
(add-to-list 'org-latex-classes
             '("memoir"
               "\\documentclass[12pt,a4paper,article]{memoir}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))

;; Reconfigure memoir to make a book (or report) from a org subtree
(add-to-list 'org-latex-classes
             '("section-to-book"
               "\\documentclass{memoir}"
          ("\\chapter{%s}" . "\\chapter*{%s}") ;; actually: BOOK TITLE!
          ("\\section{%s}" . "\\section*{%s}") ;; actually: Chapter!
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")))

;; Letter
(add-to-list 'org-latex-classes
             '("letter"
               "\\documentclass{letter}"
               ;; Should not use subsections at all!:
               ("\\chapter{%s}" . "\\chapter*{%s}")
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")))

(add-to-list 'org-latex-classes
             '("newlfm-letter"
            "\\documentclass[11pt,letter,dateno,sigleft]{newlfm}"
               ;; Should not use subsections at all!:
               ("\\chapter{%s}" . "\\chapter*{%s}")
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")))
```

11.1 export subtree as latex with header selected from file template

Todo:

11.1.1 TODO Remodel key commands:

- H-h H-e: Export latex as pdf and open

  No argument ask for template

  1-u = '(4) repeat previous file/subtree/template selection

- H-h H-E: Export latex into buffer and open window

  No argument ask for template

  1-u = '(4) repeat previous file/subtree/template selection

11.1.2 TODO Define command for switching between xelatex/pdflatex

11.1.3 The code

```
(defvar latex-templates-path
  (file-truename "~/Dropbox/000WORKFILES/SNIPPETS_AND_TEMPLATES"))

(defvar latex-section-template
  '(("\\section\{%s\}" . "\\section*\{%s\}")
    ("\\subsection\{%s\}" . "\\subsection*\{%s\}")
    ("\\subsubsect1on\{%s\}" . "\\subsubsection*\{%s\}")))

(defvar org-latex-last-chosen-file-name)

(defun org-export-subtree-as-latex-with-header-from-file (&optional use-previous-setting-p)
  (interactive "P")
 (org-latex-export-with-file-template t use-previous-setting-p t))

(defun org-export-subtree-as-pdf-with-header-from-file (&optional use-previous-setting-p)
  (interactive "P")
 (org-latex-export-with-file-template nil use-previous-setting-p t))

(defun org-export-buffer-as-latex-with-header-from-file (&optional use-previous-setting-p)
  (interactive "P")
 (org-latex-export-with-file-template t use-previous-setting-p nil))

(defun org-export-buffer-as-pdf-with-header-from-file (&optional use-previous-setting-p)
  (interactive "P")
 (org-latex-export-with-file-template nil use-previous-setting-p nil))
```

```
(defun org-latex-export-with-file-template (&optional as-latex-buffer-p use-previous-setting-
 (let* (;; backup to restore original latex-classes after this operation:
        (org-latex-classes-backup org-latex-classes)
       (paths (file-expand-wildcards (concat latex-templates-path "/*.tex")))
        (names-and-paths
         (mapcar
          (lambda (x)
         (cons (file-name-sans-extension (file-name-nondirectory x)) x))
          paths))
        (menu (grizzl-make-index (mapcar 'car names-and-paths)))
        (chosen-filename
      (if (and use-previous-setting-p org-latex-last-chosen-file-name)
              org-latex-last-chosen-file-name
         (grizzl-completing-read "Choose latex template: " menu)))
       (chosen-template-path (cdr (assoc chosen-filename names-and-paths)))
        (this-buffers-latex-class
      (plist-get (org-export-get-environment 'latex t nil) :latex-class))
        latex-header
        (latex-sections
       (or (cddr (assoc this-buffers-latex-class org-latex-classes))
              latex-section-templates)))
    (when chosen-template-path
      (setq org-latex-last-chosen-file-name chosen-filename)
      (setq latex-header
            (with-temp-buffer
              (insert-file-contents chosen-template-path)
              (concat
               "[NO-DEFAULT-PACKAGES]\n"
               "[NO-EXTRA]\n"
               "\n"
               (buffer-string))))
      ;; Create custom org-latex-classes to use this template:
      (setq org-latex-classes
            (list
             (append
              (list this-buffers-latex-class latex-header)
              latex-sections)))
      (if as-latex-buffer-p
          (org-latex-export-as-latex nil subtree-p nil nil)
      (org-open-file (org-latex-export-to-pdf nil subtree-p nil nil)))
      ;; restore original latex classes:
      (setq org-latex-classes org-latex-classes-backup)
      ;; Open the chosen template for inspection and tweaking:
    (unless (get-buffer (file-name-nondirectory chosen-template-path))
        (split-window-vertically)
```

```
        (find-file chosen-template-path)))))

(global-set-key (kbd "H-h H-e") 'org-export-subtree-as-pdf-with-header-from-file)
(global-set-key (kbd "H-h H-E") 'org-export-subtree-as-latex-with-header-from-file)
(global-set-key (kbd "H-h H-C-e") 'org-export-buffer-as-pdf-with-header-from-file)
(global-set-key (kbd "H-h H-C-E") 'org-export-buffer-as-latex-with-header-from-file)
```

12 Org-crypt: Encrypt selected org-mode entries

```
(require 'org-crypt)
(org-crypt-use-before-save-magic)
(setq org-tags-exclude-from-inheritance (quote ("crypt")))
;; GPG key to use for encryption
;; Either the Key ID or set to nil to use symmetric encryption.
(setq org-crypt-key nil)
```

13 org-reveal, ox-impress: Export slides for Reveal.js and impress.js from orgmode

Load org-reveal to make slides with reveal.js
https://github.com/yjwen/org-reveal/ https://github.com/kinjo/org-impress-js.el

```
(require 'ox-reveal)
(require 'ox-impress-js)
```

14 Folding and unfolding, selecting headings

14.1 Extra shortcut: Widen

```
(eval-after-load 'org
  '(define-key org-mode-map (kbd "H-W") 'widen))
```

14.2 Macro: toggle drawer visibility for this node

See: http://stackoverflow.com/questions/5500035/set-custom-keybinding-for-specific-emacs-mode

```
(fset 'org-toggle-drawer
  (lambda (&optional arg) "Keyboard macro." (interactive "p") (kmacro-exec-ring-item (quote ([6

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c M-d") 'org-toggle-drawer))
```

14.3 Toggle folding of current item (Command and keyboard command)

```
(defun org-cycle-current-entry ()
  "toggle visibility of current entry from within the entry."
  (interactive)
  (save-excursion)
  (outline-back-to-heading)
  (org-cycle))

(eval-after-load 'org
 '(define-key org-mode-map (kbd "C-c C-/") 'org-cycle-current-entry))
```

14.4 Keyboard Command Shortcut: Select heading of this node (for editing)

Note: outline-previous-heading (C-c p) places the point at the beginning of the heading line. To edit the heading, one has to go past the * that mark the heading. org-select heading places the mark at the beginning of the heading text and selects the heading, so one can start editing the heading right away.

```
(defun org-select-heading ()
  "Go to heading of current node, select heading."
  (interactive)
  (outline-previous-heading)
 (search-forward (plist-get (cadr (org-element-at-point)) :raw-value))
  (set-mark (point))

  (beginning-of-line)
  (search-forward " "))

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c C-h") 'org-select-heading))
```

15 Encryption

```
(require 'org-crypt)
(org-crypt-use-before-save-magic)
(setq org-tags-exclude-from-inheritance (quote ("crypt")))
;; GPG key to use for encryption
;; Either the Key ID or set to nil to use symmetric encryption.
(setq org-crypt-key nil)
```

16 Create menu for org-mode entries (lacarte lets you reach it from the keyboard, too)

```
(add-hook 'org-mode-hook
          (lambda () (imenu-add-to-menubar "Imenu")))
(setq org-imenu-depth 3)
```

17 Property shortcuts for collaboration: From-To

Note: searchable both with org-mode match: C-c / p and with icicles search, org-icicle-occur or icicle-occur, here: C-c C-'

```
(defun org-from ()
  "Set property 'FROM'."
  (interactive)
 (org-set-property "FROM" (ido-completing-read "From whom? " '("ab" "iz"))))

(defun org-to ()
  "Set property 'TO'."
  (interactive)
 (org-set-property "TO" (ido-completing-read "To whom? " '("ab" "iz"))))

(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c x f") 'org-from))
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c x t") 'org-to))
```

18 OBSOLETE fname-find-file-standardized: Consistent multi-component filenames

```
(defvar fname-parts-1-2 nil)
(defvar fname-part-3 nil)
(defvar fname-root "~/Dropbox/000Workfiles/2014/")
(defvar fname-filename-components
  (concat fname-root  "00000fname-filename-components.org"))

(defun fname-find-file-standardized (&optional do-not-update-timestamp)
  (interactive "P")
  (unless fname-part-3 (fname-load-file-components))
  (setq *grizzl-read-max-results* 40)
  (let* ((root fname-root)
         (index-1 (grizzl-make-index
                   (mapcar 'car fname-parts-1-2)))
         (name-1 (grizzl-completing-read "Part 1: " index-1))
```

```
      (index-2 (grizzl-make-index (cdr (assoc name-1 fname-parts-1-2))))
         (name-2 (grizzl-completing-read "Part 2: " index-2))
         (index-3 (grizzl-make-index fname-part-3))
         (name-3 (grizzl-completing-read "Part 3: " index-3))
         (path (concat root name-1 "_" name-2 "_" name-3 "_"))
         (candidates (file-expand-wildcards (concat path "*")))
         extension-index extension final-choice)
    (setq final-choice
       (completing-read "Choose file or enter last component: " candidates))
    (cond ((string-match (concat "^" path) final-choice)
           (setq path final-choice))
      (t
       (setq extension (ido-completing-read
                "Enter extension:" '("org" "el" "html" "scd" "sc" "ck")))
       (setq path (concat path final-choice
                  (format-time-string "_%Y-%m-%d-%H-%M" (current-time))
                           "." extension))))
    (find-file path)
    (unless do-not-update-timestamp
     (set-visited-file-name
      (replace-regexp-in-string
     "_[0-9]\\{4\\}-[0-9]\\{2\\}-[0-9]\\{2\\}-[0-9]\\{2\\}-[0-9]\\{2\\}"
      (format-time-string "_%Y-%m-%d-%H-%M" (current-time)) path)))
    (kill-new (buffer-file-name (current-buffer)))))

(defun fname-load-file-components (&optional keep-buffer)
  (interactive "P")
  (let ((buffer (find-file fname-filename-components)))
    (fname-load-file-components-from-buffer buffer)
    (unless keep-buffer (kill-buffer buffer)))
  (message "file component list updated"))

(defun fname-load-file-components-from-buffer (buffer)
  (set-buffer buffer)
  (setq fname-parts-1-2 nil)
  (setq fname-part-3 nil)
  (org-map-entries
   (lambda ()
     (let ((plist (cadr (org-element-at-point))))
       (cond
        ((equal (plist-get plist :level) 2)
         (setq fname-parts-1-2
               (append fname-parts-1-2
                       (list (list (plist-get plist :raw-value))))))
        ((equal (plist-get plist :level) 3)
         (setcdr (car (last fname-parts-1-2))
```

```
                (append (cdar (last fname-parts-1-2))
                        (list (plist-get plist :raw-value))))))))
   "LEVELS1_2")
  (org-map-entries
   (lambda ()
     (let ((plist (cadr (org-element-at-point))))
       (when
           (equal 2 (plist-get plist :level))
         (setq fname-part-3
          (append fname-part-3 (list (plist-get plist :raw-value)))))))
   "LEVEL3"))

(defun fname-edit-file-components ()
  (interactive)
  (find-file fname-filename-components)
  (add-to-list 'write-contents-functions
               (lambda ()
           (fname-load-file-components-from-buffer (current-buffer))
           (message "Updated file name components from: %s" (current-buffer))
                (set-buffer-modified-p nil)))
  ;; Debugging:
  (message "write-contents-functions of file %s are: %s"
           (buffer-file-name) write-contents-functions))
  (defun fname-menu ()
  (interactive)
  (let ((action (ido-completing-read
                 "Choose action: "
                 '("fname-edit-file-components"
                  "fname-load-file-components"
                  "fname-find-file-standardized"))))
    (funcall (intern action))))

(global-set-key (kbd "H-f f") 'fname-find-file-standardized)
(global-set-key (kbd "H-f m") 'fname-menu)
(global-set-key (kbd "H-f e") 'fname-edit-file-components)
(global-set-key (kbd "H-f l") 'fname-load-file-components)
```

19 Macro: toggle drawer visibility for this section;

See: http://stackoverflow.com/questions/5500035/set-custom-keybinding-for-specific-emacs-mode

```
(fset 'org-toggle-drawer
  (lambda (&optional arg) "Keyboard macro." (interactive "p") (kmacro-exec-ring-item (quote ([6
```

```
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c M-d") 'org-toggle-drawer))
(eval-after-load 'org
  '(define-key org-mode-map (kbd "C-c C-'") 'org-edit-special))
```

20 org-export for docpad

```
(defun org-html-export-as-html-body-only ()
 "Export only the body. Useful for using the built-in exporter of Org mode
with the docpad website framework."
    (interactive)
    (let ((path
           (concat
            (file-name-sans-extension (buffer-file-name))
            ".html")))
      (message path)
      (org-html-export-as-html
          nil ;; async
          nil ;; subtreep
          nil ;; visible-only
          t   ;; body only
          ;; ext-plist (not given here)
          )
      (write-file path)
      (message (format "written to path: %s" path))))

(global-set-key (kbd "H-e H-b") 'org-html-export-as-html-body-only)
```

21 Internal: Load org-pm

```
(org-babel-load-file "/Users/iani/Documents/Dev/Emacs/org-publish-meta/org-pm.org")
```